"Express Mail" Mailing Label No. **EK483826605US**

PATENT APPLICATION
ATTORNEY DOCKET NO. SUN-P6876-PIP

METHOD AND APPARATUS FOR DETERMINING CLASS DEPENDENCIES OF OBJECTS AND/OR CLASSES

Inventor(s): Paul A. Lovvik and Junaid A. Saiyed

Related Application

[0001] This application hereby claims priority under 35 U.S.C. §119 to a Provisional Patent Application entitled, "Class Dependency Analyzer" filed August 10, 2001 by inventors Paul A. Lovvik and Junaid A. Saiyed (Application No. 60/311,631).

BACKGROUND

Field of the Invention

[0002] The present invention relates to computer systems and software. More specifically, the present invention relates to a method and an apparatus for determining the dependent classes of objects and/or classes.

Related Art

[0003] The advent of object-oriented programming and just-in-time compilers has brought about many changes in the way in which applications are written and distributed. Traditionally, applications are compiled for specific operating systems, and a copy must be distributed for every platform on which the application will run. With modern platform-independent languages, instead of compiling the source code, the source code is distributed and is compiled on the end machine at the time of use. This practice has many advantages with the biggest being true platform independence. Write your application once and it can be run on virtually any machine.

[0004] Object-oriented programming further revolutionized the software industry by organizing code into classes and objects, which reduced distribution sizes and further promoted reusing code. One programmer can borrow a class or an object from another programmer to add functionality to an application. Upon distribution of an application, all of the classes referenced by that application must be distributed with the application if the client system does not already contain a copy of the classes. In addition, classes can contain references to other classes which must also be included for the application to work properly.

[0005] As a direct result of this type of programming, several major problems have been created. First, in order to support application delivery over the Internet, programmers are looking to compact the size of their distribution files to reduce download times, and in turn, heighten customer satisfaction. Many programmers accomplish this by running the application on a machine with just the core classes installed and adding the necessary classes as the application reports that these classes are not found. This is very time consuming, and is not all-inclusive. If a certain part of the application is not invoked, or the system contained more than just the core classes, it is possible that some critical classes

might be missed. Some programmers simply add all of the class files that are on their machine. This leads to extremely large and unnecessary distributions.

[0006] This problem is further compounded by the fact that classes can contain references to other classes, which in turn can contain references to yet more classes, and so forth. Even if you were to document every class that is directly referenced by an application as it was being developed, you would almost certainly be missing classes critical for the execution of the application. Because of their referential nature, it becomes extremely difficult to create a list of all the classes necessary, but only the classes necessary, for an application to execute.

[0007] Simple object databases can be constructed by serializing the database objects into an indexed data file. These objects can be reconstituted by deserializing that data only if all of the supporting classes for that data are accessible. Such a database that is shared between multiple applications (a system-wide object registry, for example) can be corrupted by a client application that inserts a new object into the shared database that references a class that is not commonly available. When the object is deserialized within an application that does not have access to the classes required to reconstitute that object, a ClassNotFound type of exception or error is generated.

[0008] What is needed is a method and an apparatus for efficiently determining the dependent classes of an object and/or a class.

SUMMARY

[0009] One embodiment of the present invention provides a system that determines class dependencies of a class. The system operates by: receiving a representation of the class; creating a model of the class from the representation; analyzing the model to detect references to supporting classes; and creating a list

of all dependent classes. For each supporting class found, the system recursively determines the dependencies of that class and adds them to the list.

[0010] In one embodiment of the present invention, the system identifies classes that an object depends on by: receiving a representation of an object; serializing the referenced object; parsing the data resulting from the object serialization to identify classes referenced from the target object's properties, configuration, or state; and analyzing and documenting the class dependencies of the object and all dependent classes.

[0011] In one embodiment of the present invention, the system saves the list of dependent classes to a storage structure.

[0012] In one embodiment of the present invention, the system saves the list of dependent classes to a hash table or a database.

[0013] In one embodiment of the present invention, the system creates a distribution list or a distribution file that facilitates the distribution of a program.

[0014] In one embodiment of the present invention, the system inserts an object into an object database. Upon insertion into the database, the system determines if all of the dependent classes are present in the class path of the system containing the database. If any classes are not present, the system copies them into the class path.

[0015] In one embodiment of the present invention, the system filters the list of dependent classes to remove all duplicate and core classes.

[0016] In one embodiment of the present invention, the system saves the list of all dependent classes to cache to facilitate subsequent determinations of dependent classes.

BRIEF DESCRIPTION OF THE FIGURES

[0017] FIG. 1 illustrates a computer system with a class dependency analyzer in accordance with an embodiment of the present invention.

[0018] FIG. 2 illustrates the class dependency analyzer in accordance with an embodiment of the present invention.

[0019] FIG. 3 illustrates the cache saved within the class dependency analyzer for a class in accordance with an embodiment of the present invention.

[0020] FIG. 4 illustrates the results returned by the class dependency analyzer for a class in accordance with an embodiment of the present invention.

[0021] FIG. 5 is a flowchart illustrating the process of using the class dependency analyzer to determine the dependencies of a collection of objects and classes in accordance with an embodiment of the present invention.

[0022] FIG. 6 is a flowchart illustrating the process of analyzing an object for dependent classes in accordance with an embodiment of the present invention.

[0023] FIG. 7 is a flowchart illustrating the process of analyzing a class for dependent classes in accordance with an embodiment of the present invention.

[0024] Table 1 is an example interface for the class dependency analyzer.

DETAILED DESCRIPTION

[0025] The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0026] The data structures and code described in this detailed description are typically stored on a computer readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital versatile discs or digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Computer System with a Class Dependency Analyzer

[0027] FIG. 1 illustrates a computer system with a class dependency analyzer in accordance with an embodiment of the present invention. Computer system 100 contains object 102 which exists inside of platform-independent virtual machine 104. Computer system 100 can generally include any type of computer system, including, but not limited to, a computer system based on a microprocessor, a mainframe computer, a digital signal processor, a portable computing device, a personal organizer, a device controller, and a computational engine within an appliance.

[0028] Platform-independent virtual machine 104 also contains object database 106. Object database 106 can include any type of system for storing objects in non-volatile storage. This includes, but is not limited to, systems based upon magnetic, optical, and magneto-optical storage devices, as well as storage devices based on flash memory and/or battery-backed up memory.

[0029] Platform-independent virtual machine 104 adds object 102 to object database 106. Object database 106 includes storage 108 and class dependency analyzer 110. Upon the insertion of a new object into storage 108 in

object database 106, class dependency analyzer 110 analyzes object 102 for references to classes on which object 102 depends. If a class is discovered, in this case class 112, class dependency analyzer 110 analyzes the discovered class 112 for references to classes on which class 112 depends. This process continues until no more classes are discovered. Upon completion, class dependency analyzer 110 checks class path 114, which resides in file system 116, for the presence of all of the classes detected when analyzing object 102. If the classes that were detected when analyzing object 102 are not present in class path 114, class dependency analyzer 110 adds the classes to class path 114.

[0030] Platform-independent virtual machine 104 and platform-independent virtual machine 118 can be included wholly inside of computer system 100 or can span multiple computer systems. Platform-independent virtual machine 118 retrieves object 102 from storage 108 within object database 106. Class dependency analyzer 110 analyzes object 102 for any class dependencies and retrieves them from class path 114 if necessary

Class Dependency Analyzer

[0031] FIG. 2 illustrates the class dependency analyzer in accordance with an embodiment of the present invention. Class dependency analyzer 110 contains object analyzer 200 and class analyzer 202. When the add object process 203 is invoked, a representation of the object is sent to object analyzer 200. Object analyzer 200 analyzes the object and determines all of the classes that the object references. Object Analyzer 200 then passes the list of dependent classes 204 to class analyzer 202.

[0032] Class analyzer 202 takes the list of dependent classes 204 from object analyzer 200 as well as any class names passed in directly from the add class name process 206. Class analyzer 202 analyzes the classes one at a time by

checking cache 210 to see if the class has been analyzed previously, and if not, analyzing a representation of the class for dependent classes. Once the representation has been analyzed, the list of dependent classes including the name of the analyzed class is stored in result 208, as well as in cache 210 for subsequent lookups. The resulting list of dependent classes is returned via the get class names process 212.

[0033] An example interface for class dependency analyzer 110 is as follows:

ClassDependencyAnalyzer

public void setClassnameFilter(ClassnameFilter filter);
public void reset();
public void addObject(Object o);
public void addClassname(String classname);
public Enumeration getClassnames();

Table 1

Cache Saved within the Class Dependency Analyzer

[0034] FIG. 3 illustrates the cache saved within the class dependency analyzer for a class in accordance with an embodiment of the present invention. Cache 210 contains a set of class names with lists of all of the dependent classes for the specified class as were determined in prior analysis by the class dependency analyzer. In this embodiment of the present invention, the cache is saved in a hash table where the class name is used as a key to lookup the resulting list of dependent classes.

Results Returned by the Class Dependency Analyzer

[0035] FIG. 4 illustrates the results returned by the class dependency analyzer for a class in accordance with an embodiment of the present invention. Result 208 contains the name of the class analyzed and a list of all of the

dependent classes discovered for the analyzed class. In this embodiment of the present invention, along with cache 210 illustrated in FIG. 3, the analyzed class name is of class java.lang.String. A subset of the resulting dependent classes includes java.lang.Object and java.lang.Integer. (The term JAVA™ is a trademark of SUN Microsystems, Inc. of Palo Alto, California.)

Process of the Class Dependency Analyzer

[0036] FIG. 5 is a flowchart illustrating the process of using the class dependency analyzer 110 to determine the dependencies of a collection of objects and classes in accordance with an embodiment of the present invention. Class dependency analyzer 110 starts by analyzing supplied objects for dependent classes (step 500). Upon completion of the object analysis process, class dependency analyzer 110 retrieves the resulting list of dependent classes from the object analysis process (step 502), as well as any additional supplied classes, and analyzes the classes for dependent classes (step 504). As the classes are analyzed, class dependency analyzer 110 gets the resulting list of dependent classes 204 (step 506) and applies a filter to remove duplicate classes and core classes (step 508). Steps 504-508 are repeated for each found class not yet analyzed until all classes all classes are analyzed.

[0037] After all classes have been analyzed, the filtered list is added to cache 210 to facilitate subsequent lookups of dependent classes (step 510). The resulting list of dependent classes with duplicate classes and core classes removed is then returned by class dependency analyzer 110 (step 512).

Process of Analyzing an Object for Dependent Classes

[0038] FIG. 6 is a flowchart illustrating the process of analyzing an object for dependent classes in accordance with an embodiment of the present invention.

The process of analyzing an object for dependent classes (step 500) starts by receiving a reference of the object (step 600). Next, the object is serialized (step 601), and the resulting data is parsed for references to classes (step 602). Note that given any object with any configuration in which various properties have been set into the object or the object has been put into a particular state, you are able to deduce only from the object's reference which classes are required to support that object. Finally, the list of classes is returned to the caller (step 604).

Process of Analyzing a Class for Dependent Classes

[0039] FIG. 7 is a flowchart illustrating the process of analyzing a class for dependent classes in accordance with an embodiment of the present invention. The process of analyzing a class for dependent classes (step 504) starts by receiving the name of a class to be analyzed (step 700). Next, the system checks cache to see if the class has been analyzed previously (step 702). If the class has been analyzed, the system retrieves the list of dependent classes from cache (step 712). If the class has not been analyzed, the system creates a model of the class (step 706) and analyzes the model for references to dependent classes (step 708). The system then returns the list of dependent classes to the caller (step 710).

[0040] The foregoing descriptions of embodiments of the present invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.